2) Develop a program to load the Iris dataset. Implement the k-Nearest Neighbors (k-NN) algorithm for classifying flowers based on their features. Split the dataset into training and testing sets and evaluate the model using metrics like accuracy and F1-score. Test it for different values of *k*(e.g., k=1,3,5) and evaluate the accuracy. Extend the k-NN algorithm to assign weights based on the distance of neighbors (e.g., *weight=1/d*^{2}). Compare the performance of weighted k-NN and regular k-NN on a synthetic or real-world dataset.

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Load the Iris dataset from a CSV file
file_path = 'IRIS.csv'  # Replace with the path to your CSV file
df = pd.read_csv(file_path)

# Separate features (X) and labels (y)
X = df.iloc[:, :-1].values  # All columns except the last one
y = df.iloc[:, -1].values   # The last column (species)

# Convert species labels to numerical values (if necessary)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Standardize the features
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Function to evaluate k-NN for different values of k
def evaluate_knn(X_train, X_test, y_train, y_test, k_values, weighted=False):
    results = {}
    for k in k_values:
        if weighted:
            knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
        else:
            knn = KNeighborsClassifier(n_neighbors=k)

        # Train the model
        knn.fit(X_train, y_train)

        # Make predictions
        y_pred = knn.predict(X_test)

        # Evaluate the model
        accuracy = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average='weighted')
        conf_matrix = confusion_matrix(y_test, y_pred)
        results[k] = {'accuracy': accuracy, 'f1_score': f1, 'conf_matrix': conf_matrix}

    return results

# Test for different values of k
k_values = [1, 3, 5]

# Evaluate regular k-NN
regular_knn_results = evaluate_knn(X_train, X_test, y_train, y_test, k_values,
weighted=False)
print("Regular k-NN Results:")
```

```
for k, metrics in regular_knn_results.items():
    print(f"\nk={k}: Accuracy={metrics['accuracy']:.4f}, F1-
Score={metrics['f1_score']:.4f}")
    print("Confusion Matrix:")
    print(metrics['conf_matrix'])

# Evaluate weighted k-NN
weighted_knn_results = evaluate_knn(X_train, X_test, y_train, y_test, k_values,
weighted=True)
print("\nWeighted k-NN Results:")
for k, metrics in weighted_knn_results.items():
    print(f"\nk={k}: Accuracy={metrics['accuracy']:.4f}, F1-
Score={metrics['f1_score']:.4f}")
    print("Confusion Matrix:")
    print(metrics['conf_matrix'])
```

Output:

```
Regular k-NN Results:

k=1: Accuracy=0.9778, F1-Score=0.9777
Confusion Matrix:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]

k=3: Accuracy=1.0000, F1-Score=1.0000
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

k=5: Accuracy=1.0000, F1-Score=1.0000
Confusion Matrix:
```

```
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

Weighted k-NN Results:

k=1: Accuracy=0.9778, F1-Score=0.9777
Confusion Matrix:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]

k=3: Accuracy=1.0000, F1-Score=1.0000
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

k=5: Accuracy=1.0000, F1-Score=1.0000
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```